SCO UNIX® System V/386

Development System

Release Notes

The Santa Cruz Operation, Inc.

SCO Document Number: 9-7-89-6.0/3.2.0D

SCO UNIX System V/386 3.2 Development System Release Notes

Release Notes
SCO UNIX® System V/386 Development System
Version 3.2
For computers running the
SCO UNIX System V/386 Operating System
September 7, 1989

1. Preface

These notes pertain to the SCO UNIX System V/386 Development System for computers running the SCO UNIX System V/386 Operating System. They contain notes on the software and documentation, and the procedure for installing the software.

We are always pleased to hear of a user's experience with our product, and recommendations of how it can be made even more useful. All written suggestions are given serious consideration.

2. Installation Notes

Note that you must have the UNIX System V/386 Operating System release 3.2 installed on your computer in order to use the UNIX System V/386 Development System release 3.2.

Before you install the Development System, make sure that you have:

- The Development System diskettes.
- The Operating System "N" volume diskettes. Depending on your installation, you are prompted to insert one or more of these diskettes as part of your Development System installation procedure.
- Your Development System serial number and activation key.

To install the Development System, you must perform the following operations:

- First, log in as root (the ''Super-User'') and bring the system to system maintenance mode.
- Enter the command: **custom** and press RETURN.
- Select the Development System installation option.
- Install the Development System as prompted.

The System V Development System contains several packages that can be installed selectively using the **custom**(ADM) utility. For example, the DOS cross development environment (linker, libraries and include files) is a distinct package that you can either install or leave out of your system.

custom(ADM) displays a complete list and short description of the packages included in the Development System and the amount of disk space needed to install each package. You can use **custom**(ADM) at any time to install or remove all or part of the Development System. Refer to the **custom**(ADM) manual page for instructions on using **custom**(ADM).

Towards the end of the installation procedure you are prompted to insert one or more of the ''N'' volumes. These diskettes are not part of the Development System distribution. (Although **custom**(ADM) may refer to them as ''Development System'' diskettes.) They are volumes from the Operating System distribution and certain Operating System dependent files and utilities must be extracted from them when you install the Development System.

Note that if you remove the Development System at any time with **custom**, you must copy the file */etc/perms/dsmd* from the Operating System distribution floppies in order to reinstall the Development System.

2.1 The CGI Development System

Included in your Development System distribution is the SCO CGI Development System software and documentation. This is a separately installable product, and instructions for installation can be found within the CGI *Release Notes*.

2.2 Packages In The Development System

The 386 Development System consists of several packages. When you install the Development System, you see a display of the available packages that resembles the following list:

Development System Packages

Name	Description	Size
ALL	Entire Development System set	31816
SOFT	Basic software development tools	14888
XNXDEV	XENIX cross development include files and utilities	152
386XDEV	XENIX 386 cross development libraries	1442
286XDEV	XENIX 286 cross development libraries	4244
LEX	Generates programs for lexical analysis	130
YACC	Yet another compiler-compiler	126
CFLOW	Generates C flow graphs	194
LINT	Syntax and usage check files and tools	566
SCCS	Source code control system	850
DOSDEV	DOS cross development libraries and utilities	2274
OS2DEV	OS/2 cross development libraries and utilities	3838
SAMPLE	Sample device drivers	228
MAN	Development System manual pages	2738

Once you have installed the 80386 Development System, please check the file */etc/default/cc* to confirm that the default settings for the compiler are those you desire.

3. General Software Notes

3.1 adb(CP)

In order to use **adb**(CP) to debug an 80286 binary, you must first change the permissions of the file */bin/x286emul*. To do this, you must log in as root (the super-user) and give the following command:

```
chmod 755 /bin/x286emul
```

adb should then work normally.

3.2 ANSI Conformance

The Microsoft C compiler and libraries included with the SCO UNIX System V/386 Development System are substantially conformant with the draft proposed ANSI C Programming Language standard X3.159-198X dated May 13, 1988.

The Development System manual pages in the *Programmer's Reference* include a section on standards conformance to provide a guide to those functions which are compliant with ANSI C, IEEE and FIPS POSIX, X/Open or SVID.

3.3 as(CP)

The pre-cmerge assembler is included with this release for those users who have programs that require it. It is called */bin/as* and is documented on the manual page **as**(CP).

3.4 AT&T C Compiler

The primary C development compiler within SCO UNIX is the Microsoft C compiler. However, for purposes of compatibility with code developed for AT&T C, the AT&T C compiler (*/bin/rcc*) is included in this release. Note that AT&T provides all modifications to this compiler and SCO adds no features to it. SCO intends to publish any pertinent information or updates available for this software as soon as provided by AT&T.

While most C code will compile without difficulty on either of these compilers, the possibility exists that some unusual C

constructs that are accepted by the AT&T compiler will not be accepted by the Microsoft compiler. Please note that the MS compiler accepts C code that conforms with the ANSI C standard, and that some ANSI C features will be rejected by the AT&T compiler.

For maximum compatibility, the following options should be used on the **rcc**(CP) command line:

-DM_XENIX
-DM_UNIX
-DM_COFF

3.5 CGI Libraries

Note that the **CGI** libraries are in OMF format and the **-xenix** option to the C compiler must be used to link these libraries.

3.6 CodeView

The **CodeView** debugger shipped with this release recognizes the following options:

-c special commands
-t sequential mode
-b monochrome display

See the *CodeView Debugger* manual for more information on **CodeView** options.

CodeView operates only on OMF format code produced by versions 3.2 and later of the Microsoft C Compiler, Linker, and Macro Assembler (**masm**) and compiled with the **-xenix**, **-xout**, or the **-x2.3** compiler options or on COFF binaries generated by **rcc**(CP).

In the **CodeView** documentation, there are special functions described that are accessed using the ALT key in conjunction with other keys. By default, these functions are not available. The file */usr/lib/keyboard/cv*, which is included in your standard distribution, must be renamed to replace the file */usr/lib/keyboard/keys* in order to access the special functions of keys used in conjunction with the ALT key. The original */usr/lib/keyboard/keys* file should

be saved as a precaution. For example, to perform the above described operation, log in as the Super-User (root) and give the following commands:

mv /usr/lib/keyboard/keys /usr/lib/keyboard/keys.bkp
cp /usr/lib/keyboard/cv /usr/lib/keyboard/keys

You should also add the command:

mapkey

to the file */etc/rc* to direct your system to customize the keyboard automatically at boot time.

However, if your system has an F12 key, the ALT key functions can be used by first pressing F12 and then the <*key*> needed. You need not make any modification to your system files if you use this method.

In this release, **CodeView** runs only on 386 COFF binaries generated by the *rcc*(CP) compiler and 386 OMF binaries generated by the **cc**(CP) compiler.

Floating point support is not yet implemented for **CodeView**.

Known problems with the **CodeView** software include:

Mouse support in **CodeView** is not available in this release.

The E command (Execute until keypress) is not yet fully implemented. If you use this feature, **CodeView** will hang indefinitely and you must terminate the program from another terminal or multiscreen.

Regular expression searching is implemented using the **regex** library, which uses a different syntax than the notation described in the *CodeView Debugger* documentation.

Programs that map the video memory, use graphics or screen switching, will cause **CodeView** to behave irregularly. However, use of the normal Multiscreens is available without difficulty.

3.7 crypt(C) and crypt(S) Libraries

The **crypt**(C) utility and **crypt**(S) libraries are not included in this release. If you are a United States resident, you can request a copy of **crypt** by calling your support center.

3.8 cxref(CP)

cxref(CP) does not correctly handle function prototypes. It's syntax analysis is based on an older version of the compiler. To work around this problem, make function prototypes subject to a conditional compilation definition. For example:

```
#ifndef NO_PROTOTYPE
int func( char *, long );
#endif
```

Then use the flag:

-DNO_PROTOTYPE

when using **cxref**.

Note that **cref**(CP) and **xref**(CP) are no longer included with the software. **cxref**(CP) provides the functionality of both of these utilities.

3.9 Device Driver Notes

The following two notes apply only to device driver developers.

3.9.1 geteblk(K)

Note that geteblk(K) does not clear the buffer it obtains. It is the responsibility of the device driver writer to clear the buffer with **clrbuf**(K) before using it. The **clrbuf**(K) manual page is included with these notes.

3.9.2 Signals

It is the responsibility of the programmer to verify that the process that is the object of a pointer is valid. Any signals used during poll time or interrupt time should check the process table to be sure that the intended process is still running and available.

3.9.3 timeout(K)

The **timeout**(K) manual page incorrectly states that **timeout**() should only be called at task time. However, some restrictions do apply if you wish to use **timeout**() during interrupt time or during an initialization routine. These restrictions are discussed in the **timeout**(K) manual page.

3.10 help(CP)

The **help**(CP) package has been modified in this release. **help** provides explanation of SCCS error messages.

3.11 ld(CP)

The **-r** option of **ld**(CP) does not work correctly for object files that contain the additional symbol table information used by the symbolic debugger. If you use the **-r** option of **ld** to partially link together several object modules that contain symbolic debugging information, the code, data, and relocation information in the resulting object module will be correct, but the symbolic debugging information will be unusable.

Also, note that you may not mix libraries of different formats with the standard **ld**(CP) command. For example, if you wish to include both OMF and COFF libraries in your program, you may not use the standard linker. You must directly invoke the XENIX linker, which is located in the */lib/xout* directory and called *ld*. The XENIX linker allows you to mix libraries of different formats.

3.12 nm(CP)

The **-n** option of **nm**(CP) generates an error with long name lists. To sort a long name list, use the command line:

nm *executable* | sort

instead of the **-n** option.

3.13 OMF/COFF Utilities

The following utilities determine whether a subject file is of OMF or COFF format and execute different binaries in each case:

ld
ar
nm
size
strip

Therefore, changing or moving any file that these utilities depend upon will cause an error. You will see the message:

can't exec <*filename*>

3.14 sccs(CP) Line Size

There is a limit of 508 characters on any physical line of any file placed under **sccs**. Writing a changed file with a line containing more than 508 characters results in corruption of the file.

3.15 sdb(CP)

sdb currently works only on COFF binaries generated by the **rcc**(CP) compiler.

Programs that are to be debugged with **sdb** must be compiled and linked with the following options:

rcc -Zi or **rcc -g**

ld -g

The following features of **sdb** that are described in the documentation are not supported in this release:

- Pattern matching for function and variable names. For example,

  x*/

  should display the values of all variables with names beginning with "x".

- The ''.'' command to redisplay the value of the last variable typed.

- The command:

  * *<file*

  to read **sdb** commands in from a file works correctly only when there is no space placed between the ''<'' character and the file name.

3.16 XENIX Software Products

When installing XENIX products on your System V/386 system, note that the libraries and include files provided are for producing x.out format binaries only. Therefore, until other products are updated for 3.2, you should use the **-xout**, **-x2.3** or the **-xenix** options when compiling using older XENIX products.

4. C Language Notes

4.1 audit(S)

Note that if you use **audit**(S), you must compile your program with the **-laudit** option to the C compiler.

4.2 brkctl(S) Library

System V/386 does not support 386 processes which have more than one data segment. For compatibility, a special library (*/lib/386/Slibbrkctl.a*) has been provided that maps **brkctl**() functions into calls to **sbrk**(S). This provides for the most common uses of **brkctl**(S), such as the allocation of additional memory.

Programs which rely on allocating multiple data segments and manipulating the sizes of those segments will need to be altered to work under System V/386.

The following describes the functionality implemented by the 386 *libbrkctl.a*. Note in particular that the 386 **brkctl**(S) returns a near pointer and that the third argument is also a near pointer. If you do not wish to alter your source code when compiling for the 386 you should include **-Dfar=** on the **cc**(CP) command line. This will

cause the preprocessor to remove all "far" keywords from your code.

```
#include <sys/brk.h>
char *brkctl(cmd, inc, ptr)
int cmd;
long inc;
char *ptr;
```

When called from 386 processes, **brkctl**() does not cause the allocation or de-allocation of far data segments. It can only be used to increase or decrease the memory allocation in the single data segment available to 386 processes.

Any of the commands **BR_IMPSEG**, **BR_ARGSEG** or **BR_NEWSEG** can be used, as they all have the same effect.

The *ptr* argument in the above program example is ignored.

It is unusual to allow the use of **BR_NEWSEG** with 386 processes since it is not possible to allocate additional data segments. However, since **BR_NEWSEG** is commonly used in small and middle model 86/286 processes to allocate additional memory, it was decided to allow **BR_NEWSEG**, but to make its functionality the same as **BR_IMPSEG**.

In order to link a 386 binary with this version of **brkctl**() you should specify **-lbrkctl** on the **cc**(CP) command line.

4.3 cc(CP) Defaults

The default code generation model for the 80386 is **-M3e**. (80386 with non-ANSI extensions enabled.) This default can be changed by editing */etc/default/cc*. See the **cc**(CP) manual page for details.

4.4 COFF File Manipulation Utilities

You must include the **-lld** flag to the C compiler when you use the system calls provided to allow manipulation of COFF files. The following system calls need this flag:

ldahread(S)
ldclose(S)
ldfhread(S)
ldgetname(S)
ldlread(S)
ldlseek(S)
ldohseek(S)
ldopen(S)
ldrseek(S)
ldshread(S)
ldsseek(S)
ldtbindex(S)
ldtbread(S)
ldtbseek(S)

4.5 directory(S)

There are two versions of the standard directory services. One version is found in **libc** and the other is found in **libdir**. For convenience, the following table shows the correct syntax for each version:

library	struct name	compiler option	include file
libc.a	dirent		<dirent.h>
libdir.a	direct	**-ldir**	<sys/ndir.h>

4.6 International Development

These notes describe enhancements provided by the SCO International Development System

4.6.1 conv(S) Routines

Two macros, **todigit** and **toint**, have been added. The **todigit** macro returns the digit character associated with the integers 0-9. The **toint** macro does the opposite.

The macro definitions of **toupper** and **tolower** have been replaced by library functions. This is to introduce behavior that is a superset of the ANSI standard and equivalent to SVID and X/OPEN.

The arguments to **toupper** and **tolower** are converted to integers. For this reason, care should be taken that character arguments are supplied as unsigned characters to avoid problems with sign-extension of 8-bit character values.

4.6.2 ctype(S) Routines

Several macros have been revised to accommodate 8-bit characters for international development. While **isascii** is defined on all integer values, the rest are defined on integers, the value of which can be represented as an unsigned char, or is equal to the single non-character value EOF. For more information, see the *International Development Guide*.

4.6.3 Limitations

Although the underlying libraries used by the Development System and other utilities have been modified to function with 8-bit characters, only a key number of utilities have been verified to function properly in all cases. Unless the utility or routine is listed as verified in these notes, data files and keyboard input should be restricted to 7-bit ASCII characters. For example, in the case of the C compiler, 8-bit characters can be used in character and string literals. However, for reasons of compatibility with other utilities and C compilers on other systems, it is important to confine use of 8-bit characters to data files, and use *\nnn* escapes to specify 8-bit characters within C source.

4.6.4 Native Language Support

A number of routines have been introduced to ensure compatibility with the X/OPEN Portability Guide. The routines are **nl_init**, **nl_cxtime**, **nl_printf**, **nl_langinfo**, **nl_scanf** and **nl_strcmp**.

The **nl_init** routine is the X/OPEN equivalent of **setlocale**.

4.6.5 New or Changed Library Features

This section describes new or enhanced features provided by release 2.0 of the Development System International Supplement. New routines have been introduced that make it possible to use locale-specific information, such as character set and time format selection in programs.

Refer to the *International Development Guide* and manual pages for more information on the routines that follow.

4.6.6 Routines for Handling Regular Expressions

The **regexp** and **regex** routines have been verified for 8-bit operation. The **regexp** syntax has been extended to accommodate 8-bit characters. Refer to the **ed**(C) manual page for more information.

As part of the modifications for 8-bit operation, the amount of storage required for character ranges in the compiled expression may be double that of the standard requirement.

4.6.7 The setlocale(S) Routine

The **setlocale** routine is used to read or set the international environment according to a specified category and locale.

At program startup the equivalent of the following call is executed:

```
setlocale(LC_ALL, "")
```

This is so that the initial locale for the program is defined as in the user environment.

4.6.8 The strftime(S) Routine

This routine converts and formats date and time information used by applications.

4.6.9 String Collation

String collating routines have been introduced to accommodate 8-bit character sets.

4.7 irand48(S) and krand48(S)

The functions **irand48**(S) and **krand48**(S) are not supported in this release.

4.8 Longest Allowed Path Names

The longest path name in a C program is restricted to 1024 bytes. System calls that require path names as arguments will now fail, setting **errno** to **ENAMETOOLONG**, if a longer path name is given.

Previously, the path name was not restricted by the operating system; however, most programs gave an ad hoc limit to the length. Generally, these limits were well below 1024 bytes, so most programs should not be affected by this change.

Note also that any path component longer than 14 characters also causes **ENAMETOOLONG** to be set.

The **limits.h** file defines a constant **_POSIX_PATH_MAX** to be the longest length of a path name. Use *pathconf*() to determine **_POSIX_PATH_MAX** portably over various FFS filesystem types. In previous releases, this file incorrectly set the macro to 256, but it will probably be changed in a future release to 1024. Local system administrators can safely change the value for **_POSIX_PATH_MAX** to 1024 without harm, since the Release 3.1 and later systems internally use the longer limit.

You are encouraged to include the *limits.h* file with a statement like

```
#include <limits.h>
```

and to refer to **_POSIX_PATH_MAX** for the longest path name allowed.

4.9 malloc Issues

There are two versions of **malloc** distributed with the Development System. The standard **malloc** is contained in the file */lib/libc.a*. There is an alternate **malloc** in */lib/libmalloc.a*. Both are documented under the **malloc**(S) manual page.

If your program uses many **malloc** and **free** calls, running the program under the System V/386 Operating System may cause an excessive page swapping problem as **malloc** must search the entire list of allocated and free blocks. If you are using **malloc** and **free** calls extensively, it is suggested that you use the alternate **malloc** found in */lib/libmalloc.a*. To use */lib/libmalloc.a*, compile your program with the **-lmalloc** flag on your **cc**(CP) command line.

The alternate **malloc** maintains a separate list of free blocks and may be faster, but data in any allocated block is immediately overwritten when the block is declared free. The standard **malloc** preserves data between consecutive **free** and **malloc** calls. Some programs rely on this functionality of the standard **malloc**.

4.9.1 mallinfo()

Note that you must call **malloc(S)** at least once before calling **mallinfo(S)**. Otherwise, calling **mallinfo()** may result in unpredictable behavior. Note that **mallinfo()** is only available within large model **-lmalloc**.

4.10 Memory Models

The only memory model supported for 386 code is "small" model. Small model has two 32 bit segments; one for code and one for data. Small, middle, large, huge, and compact models are supported for 8086/286 code.

4.11 MINDOUBLE

The C compiler does not compute the value of the constant MINDOUBLE correctly. It always evaluates to 0 when used in expressions. This constant is defined in */usr/include/values.h*.

4.12 -pack Option

Note that the **-pack** option to **cc**(CP) is not supported for programs linked with SCO-distributed libraries.

4.13 passlen(S)

Note that you must use the **-lm** option to the C compiler if you use the **passlen**(S) system call.

4.14 Preprocessor Names

Symbols which are used in **#define**, **#ifdef** and other preprocessor commands must now conform with the same rules as those for C identifiers. They must begin with an alphabetic character or an underscore and may only contain alphanumeric characters and the underscore character.

Previous versions of the C compiler allowed other non-alphanumeric characters such as "." in preprocessor symbols.

4.15 Protected Database Functions

Programs that manipulate the various 3.2 protected databases must be compiled with the **-lprot** option to the C compiler. Typically, these programs include the header file *<prot.h>*.

4.16 raise(S)

Note that you must compile with the **-lx** option to the C compiler if you use the **raise**(S) system call.

4.17 Semaphores in 80286 Code

To use semaphores in code that is to be compiled as an 80286 binary, you must make the following change in <sys/types.h>:

```
typedef int key_t;
```

must be changed to:

```
typedef long key_t;
```

4.18 sputl(S)

Note that you must compile with the **-lld** option to the C compiler if you use the **sputl**(S) system call.

4.19 Stack Size

80386 programs have a variable sized stack that is expanded by the kernel as needed. 8086/286 programs run under the 386 Operating System have a fixed size stack. The default stack size for 8086/286 binaries is 4 Kilobytes unless the **-F** option of the linker was used. Once compiled, you can change the stack size of an 8086/286 program using the **-F** option to **fixhdr**(C).

4.20 Stricter Checking of Storage Class in Declarations

The ANSI standard requires that declarations and definitions of functions and variables must have matching storage classes as well as matching types. Typically, this causes problems in code when a function is first used (and implicitly declares it as an "extern int") and is later defined as "static int." ANSI requires that the first use of the function be preceded by an explicit declaration. For example:

```
static int foo(); /*required by ANSI C */

Program text

        foo();

static foo()
```

The **-Me** option disables this strict checking of storage class.

4.21 String Constants

The 386 C compiler has been extended to allow 4K bytes in string constants. The compiler generates the following message if this limit is exceeded:

```
string too big, trailing characters truncated.
```

4.22 Substituting Parameters in Quoted Strings

The proposed ANSI standard for C does not allow the substitution of macro formal parameters within quoted strings. However, many existing C compilers allow this. Consider the following macro definition and use:

```
#define CTRL(c)          ('c' & 0x1f)
putchar(CTRL(g));
```

The above statement operates as intended when using the AT&T Compiler. The AT&T compiler expands this to:

```
putchar(('g' & 0x1f));
```

But according to the ANSI standard it must be:

```
putchar(('c' & 0x1f));
```

The Microsoft C compiler follows the ANSI standard strictly, therefore you must use this substitution as follows:

```
#define CTRL(c)          (c & 0x1f)
putchar(CTRL('g'));
```

Then the compiler expands this to:

```
putchar(('g' & 0x1f));
```

The **-Me** option to the 386 C compiler allows the non-ANSI functionality described above, but displays a warning that a non-standard extension has been used if the warning level is set to 2 or higher.

4.23 terminfo curses

When you use terminfo curses in a 286 binary, you should compile with the **-F** option set to 2000 or greater to increase the fixed stack size. For small model programs, compile with the **-i** flag.

4.24 Third Party Language Library

If you are using a third party language product, such as Microsoft Pascal, you may experience errors when using the link editor supplied with the language product. This is due to the addition of a new library that is necessary for linking modules with C libraries. This library is called **/lib/libh.a** and needs to be added explicitly to your link command in order to correctly compile your programs.

4.25 Tokens Following #else and #endif

The C compiler issues a warning if it finds anything other than blank space or comments following an **#else** or **#endif** preprocessor directive. Thus the first example below would produce a warning but the second would not.

```
#endif  M_I386
#endif  /* M_I386 */
```

These warnings are produced at the default warning level of 1 on both the 286 and 386 compilers.

4.26 Transforming Formal Parameters to Strings

The ANSI standard defines a mechanism for transforming macro formal parameters into quoted strings (the stringizing operator, ''#'') and an operator for pasting strings, ''##''. These have been added to both the 286 and 386 preprocessors. A slight restriction on the 386 is that white space is not allowed between the stringizing operator and the formal parameter it operates on.

4.27 types.h

Some of the C language *.h* files require that *<sys/types.h>* be included first. An error message generally occurs when a type is used in an #include file but not declared. Often, the problem is corrected by including *<sys/types.h>* earlier in the program.

4.28 Variable Assignments

Assignments of the following form generate incorrect code when compiled with optimization enabled:

```
int i;
register  char *buf;

buf[ i + i ] = buf[ i ];
buf[ i ] = buf[ i + i ];
```

Note that if ''buf'' is not a register variable, or if the index expression ''i + i'' contains any term other than ''i'', such as a constant or another variable, correct code is generated. Thus:

```
buf[ i + i + 1 ] = buf[ i ];
buf[ i + n ]     = buf[ i ];
```

both work correctly.

4.29 Variable Declarations

The error: *Segmentation Violation: core dumped* may occur at compile time if you declare too many variables within a single declaration statement. For example, the following text may cause the compiler to dump core:

```
char *
        x1,     /* comment */
        x2,     /* more comment */
        .
        .
        .
        x934;   /* more comment */
```

Twenty five variables in a declaration is a safe maximum. Break longer declarations into two or more statements to avoid this possible problem.

4.30 XENIX Cross Development Notes

The notes in this section refer to points of interest to C programmers developing for the XENIX environment only.

4.30.1 286 Floating Point Emulation

286 floating point emulation software in 286 Operating Systems prior to 2.3 does not correctly execute floating point comparisons between the top of the floating point stack and memory. When you develop applications using the **-dos**, or the **-M0**, **-M1**, or **-M2** flags, you should use the **-Go** option to **cc**(CP) to instruct the compiler to allow for this. Note that the **-compat** flag implies the **-Go** option automatically.

When using masm, special care needs to be taken not to make floating comparisons with memory operands. To execute correctly on previous 286 floating point emulators, only comparisons of

different stack elements should be made. Note that pushing a memory operand onto the stack causes the source/destination relation between the two operands to be reversed in a subsequent comparison operation. As an example:

```
fcom    memory_address
jg      label
```

should be written as:

```
fld     memory_address
fcomp   ST(1)
jl      label
```

4.30.2 Developing XENIX Device Drivers

To develop installable device drivers suitable for use under XENIX, you must have the XENIX Development packages installed.

4.30.3 setvbuf(S)

The order of the parameters to **setvbuf**(S) has been changed to conform with the SVID. In releases of the XENIX Operating System prior to 2.3, the order of the parameters was:

```
int setvbuf (stream, type, buf, size)
```

In release 2.3 and later of XENIX and all releases of SCO System V, the order of the parameters is:

```
int setvbuf (stream, buf, type, size)
```

The files */lib/*[LMCS]*setvbuf.o* can be used for backwards compatibility.

5. Documentation Notes

5.1 clrbuf(K) Manual Page

The clrbuf(K) manual page is included with these notes. Note that this function is kernel specific and should only be used within device drivers.

5.2 cref(CP) and xref(CP)

Note that **cref**(CP) and **xref**(CP) are no longer included with the software. **cxref**(CP) provides the functionality of both of these utilities. The **cref**(CP) and **xref**(CP) manual pages are still included in the documentation but will be removed in future releases.

5.3 directory(S) Manual Page

The functions described in the **directory**(S) manual page must be compiled with the **-ldir** flag to **cc**(CP) if the executable format is to be OMF.

5.4 fieldtype(S) Manual Page

The **fieldtype**(S) manual page refers to the function **link_fieldtyp**. The correct name of this function is **link_fieldtype**.

5.5 mknod(S) Manual Page

There is an error in the **mknod**(S) manual page. The description states that file type bits may be zero in an ordinary file, but this is not the case. If the file type bits are set to zero, this indicates that the inode is out of service.

5.6 panel(S) Manual Page

There is an error in the Syntax section of the **panel**(S) manual page. The declaration:

```
int *set_panel_userptr(panel, ptr);
```

should read:

```
int set_panel_userptr(panel, ptr);
```

5.7 setgroups(S) Manual Page

The syntax given in the **setgroups**(S) manual page is incorrect. The correct syntax is:

```
#include <sys/types.h>

int setgroups(gidsetsize, gidset)
int gidsetsize;
gid_t gidset;
```

5.8 sigset(S) manual page

The syntax given in the **sigset**(S) manual page is partially incorrect. The correct syntax for the following routines is:

```
int sigaddset (set, signo)
sigset_t *set;
int signo;

int sigdelset (set, signo)
sigset_t *set;
int signo;

int sigismember (set, signo)
sigset_t *set
int signo;
```

5.9 STREAMS Manual Pages

The following commands have features or references related to STREAMS. Additionally, The (NSL) and (STR) sets of manual pages are included in the streams documentation.

exit (S)
fcntl (S)
getmsg (S)
ioctl (S)
open (S)
poll (S)
putmsg (S)
read (S)
signal (S)
sigset (S)
write (S)

5.10 termcap(S) manual page

The **termcap**(S) manual page refers to the **stty**(S) manual page. This reference should be to **stty**(C). There is no **stty**(S) manual page.

6. POSIX Implementation Defined Issues

Null pathnames are considered invalid and generate an error. (Rev. POSIX FIPS) p. 36

chown(S) is restricted to a process with appropriate privileges. C2 tuning may restrict all cases of chown to root. (Rev. FIPS)

Only a user with appropriate privileges may set file timestamps to arbitrary values using **utime**(S). (Rev. FIPS)

System V/386 supports a value of NGROUPS_MAX greater than or equal to 8. (Rev FIPS)

chown(S) restricts changing the group-id of a file to egid or one of the supplementary group ids. (Rev. FIPS)

exec(S) saves the effective user and group id. (Rev FIPS)

kill(S) uses the saved set userid of the receiving process instead of the effective uid for determining eligibility of sending a signal. (Rev FIPS)

Upon **exit**(S) of a session process group leader, a SIGHUP is sent to each member of the session process group. (Rev FIPS)

Terminal special characters can be individually disabled using the value specified by POSIX_VDISABLE. (Rev. FIPS)

System V/386 supports POSIX_JOB_CONTROL. (Rev FIPS)

Support is provided for CPIO and USTAR data formats. (Rev FIPS)

Pathname components longer than NAME_MAX are invalid and generate an error. (Rev. FIPS)

EBUSY is supported as an errno value for **rename**(S), **rmdir**(S), and **unlink**(S). (Rev. FIPS)

ISUID and ISGID are cleared by **chown**(S), even for root process. Page 103.

Only a user with appropriate privileges may link or unlink directories. (Rev. FIPS)

If the setgid bit is set on a directory, the group ID of newly created files and directories within it is set to the group owner of the directory. (Revised POSIX FIPS).

System V/386 supports modem control on asynchronous serial terminal ports. (see the <config> parameters TERMIOS_TTY, TERMIOS_LOOP, and TERMIOS_TTY_NC.) (Rev. POSIX FIPS)

System V/386 supports the environment variable HOME for the home directory as defined in Section 2.7 of the IEEE Std. 1988-1003.1. (Revised POSIX FIPS).

System V/386 supports the environment variable LOGNAME for the login shell as defined in Section 2.7 of the IEEE Std. 1988-1003.1. (Revised POSIX FIPS).

System V/386 supports **tcgetpgrp**(S) and _POSIX_JOB_CONTROL is defined. (Revised POSIX FIPS). See the "Get Foreground Process Group ID" in the Device- and Class-Specific Functions chapter of the POSIX standard.

System V/386 supports links to directories for user with appropriate privileges. (Revised POSIX FIPS). See the "Link to a File" in the Files and Directories chapter of the POSIX standard.

Access permission is required to access an existing file. (Revised POSIX FIPS).

System V/386 does not support links between different file systems. See the "Link to a File" entry in the Files and Directories chapter of the POSIX standard.

System V/386 supports using **unlink**(S) on directories from a user with appropriate privileges. (Revised POSIX FIPS). See the "Removing Directory Entries" entry in the Files and Directories chapter of the POSIX standard.

System V/386 allows unlinking a directory when it is being used by the system or another process. See the "Remove Directory Entries" entry in the Files and Directories chapter of the POSIX standard.

System V/386 allows the removal of a directory that is being used by another process. See the "Remove a Directory" section in the Files and Directories Chapter of the POSIX standard.

System V/386 does not require write permission for existing directories when renaming them. See the "Rename a File" section in the Files and Directories chapter of the POSIX standard.

System V/386 supports a call to **rename**(S) when the directory named is in use by the system or another process. See the "Rename a File" section in the Files and Directories chapter of the POSIX standard.

System V/386 supports the setting of S_ISUID and S_ISGID by **chmod**(S). See the "Change File Modes" section in the Files and Directories Chapter. of the POSIX standard.

System V/386 does not support the EINVAL error for a call to **chown**(S) when an invalid "owner" argument is specified. See the "Change Owner and Group of a File" section in the Files and Directories chapter of the POSIX standard.

A call to **read**(S) or **write**(S) that is interrupted by a signal after it has successfully read or written any data will return the number of bytes read or written. (Revised POSIX FIPS). See "Input and

Output Primitives'' in the POSIX standard.

System V/386 supports character size of 5 bits, 6 bits, 7 bits, and 8 bits. See the ''Control Modes'' section in the Device and Class Specific Functions chapter of the POSIX standard.

System V/386 does not support changing the START and STOP special characters. See the ''Special Characters'' section in the Device and Class Specific Functions chapter of the POSIX standard.

System V/386 does not support the inclusion of the groupid with supplementary group IDs. See the ''Get Supplementary Group IDs'' section in the POSIX standard.

CLRBUF (K)

Syntax

```
#include "sys/buf.h"

int
clrbuf(bp)
struct buf *bp;
```

Description

This routine zeroes a buffer previously allocated during block I/O. You supply a pointer to a buffer header (**bp**) to *clrbuf*. The *clrbuf* routine then zeroes the data pointed to by the address in **bp->b.un.b_words** for the number of bytes specified in **bp->b_bcount**. *clrbuf* also sets **bp->b_resid** to zero (0). *clrbuf* does not have a return value. The **b_words**, **b_bcount**, and **b_resid** fields are members of the *buf* structure and are defined in the **buf.h** header file.

Call the *geteblk*(K) routine before calling *clrbuf*. *geteblk* does not clear the buffer; *clrbuf* must be used to ensure that the buffer is cleared. The *geteblk* routine assigns a value to **b_bcount**. The standard way to call *clrbuf* is as follows:

```
bp = geteblk();      /* Get a buffer */
clrbuf(bp);          /* Clear it     */
```

Warning

If your driver is using a non-standard buffering scheme managed with *buf* headers, ensure that **b_bcount** is set to a value the same or less than the size of the buffer. If **b_bcount** is set to a value larger than the current buffer size, data may be destroyed. **b_bcount** is set by *geteblk* to SBUFSIZE. SBUFSIZE is described in **/sys/fs/s5param.h** and varies by file system size.

See Also

geteblk(K)